ABSTRACT
A polyhedron can be unfolded to a net, i.e., an unfolding without overlapping, by carefully cutting along the surface. If the cuts are restricted only on the edges of the polyhedron, where should the cuts be? This is called an edge-unfolding problem, which has been extensively studied in the literature for centuries. Although several promising properties have been discovered, several recent preliminary works show that no valid net exists even for certain simple non-convex polyhedra. Therefore, we propose to convexify the input polyhedron before unfolding. More specifically, we remove local concave surface features via inflation simulation. We then eliminate global concave structure features by segmenting the polyhedron to a small number of part-aware and nearly convex components. Then the net for each nearly convex component can be obtained. We further show that convexified shapes can be continuously folded and can be easily realized by a physical self-folding machine. Our experimental results show that the proposed convexification approaches can reduce the computation time by several folds.

1 Introduction
Paper crafting enables us to fabricate a target surface from one or multiple sheets of papers which are easier to manufacture and transport [1][3]. This technique can be used to design self-folding robot [4] with rigid materials that can fold itself from a flat sheet to a 3D functional shape via uniform heating [5], magnetism [6] or lighting [7]. Designing a paper craft usually involves two main foldability analysis steps. The first step is to find a 2D representation (which could be a net, a crease pattern or a developable surface) whose folded shape approximates entire or part of the mesh. We call this an instantaneous unfolding problem, since the solution will be a function that instantaneously transforms the polyhedron to the 2D representation without going through any intermediate configurations. In this paper we focus on the net representation, which is the unfolding of the polyhedron that does not contain overlaps. Finding a valid net of a given polyhedron is known to be nontrivial because a polyhedron with \(|F|\) faces can have approximately \(2^{\sqrt{|F|}}\) different unfoldings and most of them contain overlaps especially for non-convex polyhedra. Suppose that we obtained a net, the second step is to find a foldable path that transforms the net to its folded shape continuously without self-intersection. We call this a continuous folding problem. Unfortunately, only few works take continuous folding into consideration and assume that the net is always foldable. However, this assumption is mostly incorrect especially when folding non-convex patches with rigid materials.

Though the problem of finding edge unfolding for convex polyhedra is still open, heuristic methods work well in practice. Most, if not all, nets of convex polyhedra with \(|F|\) faces can be obtained in \(O(|F| \log |F|)\) time. Furthermore, the start state, i.e., the flat net and the goal state, i.e. original convex polyhedron, may often be linearly connected in its configuration space (or C-space, see Section 3.3 for definition). This property significantly reduces the path planning time to find a continuous folding motion. Figure 2 shows a convex polyhedron and its nets, one of whose folding path is a straight line in the configuration space. However, when dealing with non-convex shapes, previous...
works \cite{8,9} show that both instantaneous unfolding and continuous unfolding become significantly more challenging.

To unfold non-convex shapes, we employ mesh inflation to remove local concave features and nearly convex decomposition method to segment a mesh into several nearly convex patches. Then both instantaneous unfolding and continuous folding problems are solved for each patch separately. However, since the patch is not exact convex, no heuristic methods guarantee to find a net (and there exist some non-convex polyhedra that can not be unfolded), we employ a genetic algorithm to find nets for those nearly convex patches. Once we obtained the net, motion planning algorithm is used to find a continuous folding path. Our experimental results show that the mesh convexification makes each step several folds faster in terms of total running time than working on the original mesh alone. It also makes manufacturing and assembling easier.

Main Results. Our study in this paper shows that, nearly convex decomposition shows great advantages in both instantaneous and continuous unfolding steps, thus the requirement of exact convexity is often unnecessary. There are mainly three advantages to using nearly convex shapes for paper crafting: 1) Easier to unfold (finding a net) and fold due to its nearly convex property. 2) Approximation ability. We can obtain exactly the same shape as the original shape if we use nearly convex patches, or obtain a bounded error approximation using convex hulls of each component. 3) Easier to assemble due to its part-aware property.

2 Related Works

2.1 Polyhedra Unfolding

Unfolding the polyhedron to a 2D shape by cutting on the surface of the polyhedron has been studied extensively in the mathematical literature, here we refer interested readers to this short survey paper \cite{10}. An unfolding without overlapping is called the net of the polyhedron. For the edge unfolding case that we are interested in, Schlickenrieder \cite{11} proposed heuristics methods for unfolding a polyhedron to a net. Ghomi \cite{12} shows that every convex polyhedron admits a simple edge unfolding after an affine transformation. Straub and Prautzsch \cite{13} and Takahashi et al. \cite{8} extended \cite{11} to unfold non-convex polyhedra, however, it becomes much harder to generate a net for non-convex shapes. Thus, they either use splitting or post merging to ensure that the unfolding does not overlap while trying to minimize the number of pieces in the final unfolding. All aforementioned works generate non-overlapping unfoldings as final results, however, whether there exists a continuous folding motion that transforms the net back to its original shape is not considered in their works.

2.2 Paper Crafting via Shape Segmentation

As mentioned in the previous section, unfolding the entire mesh into a single connected piece is hard, many approaches \cite{1,3} use shape segmentation techniques to decompose the mesh into small pieces to make the unfolding problem solvable and then assem-
ble folded shapes together to obtain the approximated shape. Mi-
tani and Suzuki [1] decompose the mesh into a few patches and app-
approximate each patch with a strip, a generalized cylinder or a
developable surface. And by assembling all folded patches to
obtain an approximation of the original one. These approaches
create new surfaces whose final result are no longer the origin-
ral surface but its approximation which might be fine for some
applications, however, these methods can generate an arbitrary
number of pieces and the cuts can be at arbitrary locations which
make assembling much harder and less fun. Another main issue
is that since they are using either generalized cylinders or devel-
opable surfaces, their ability to approximate shapes is limited.

In this paper, we look into another type of shape decomposition
method called Nearly Convex Decomposition (NCD) which seg-
ments a mesh into a countrollable number (usually small) of com-
ponents that are nearly convex, and it is part-aware. Lien and
Amato [14] proposed the first algorithm to decompose a poly-
hedron into nearly convex parts. Asaﬁ et al. [15] proposed a
method that weakly decomposes a set of points. More recently,
a method called CoRiSe [16] was proposed to obtain nearly con-
 vex components by separating the convex ridges which requires
only one user parameter: concavity tolerance τ. This method
achieves competitive results (close to human segmentation) with
much less computation time compare to other approaches.

2.3 Continuous Folding

To make a physical copy of foldable shape that can be continu-
ously folded to its 3D target shape from the net, we need to find a
feasible folding path that can bring the net to its target shape
without self-intersection. Demaine et al. [17] proposed contin-
uous unfolding algorithms for all convex polyhedra. However,
their algorithms require to cut at arbitrary locations on the surface
of the polyhedra and only work on convex shapes. These limi-
tations make it hard to be applied in practice. Tachi [18] and Xi
and Lien [19] proposed numerical methods for finding the fold-
ing/unfolding motion for rigid origami. However, these methods
either do not consider self-intersection or take advantage of the
symmetry property of the crease pattern to reduce the dimension-
ality of the configuration space. Traditional rigid origami (sin-
gle piece of rectangular or convex paper without any cuts inside)
is known to be more constrained (even over-constrained) which
usually have only one degree of freedom (DOF), thus these meth-
ods can not be directly applied to unfold polyhedra whose net
can be regarded as a tree-structure rigid origami. Song and Am-
ato [20] studied a problem of net folding, and proposed a PRM-
based planner to plan folding motion for tree-structure origami,
however, their works mainly take the advantage of the linear con-
nectable property of the C-Space (e.g., the start state and goal
state can be connected by a straight line in C-Space). Therefore,
their method does not scale to large and complicated nets whose
start state and goal state cannot be connected directly. Xi and
Lien [20] proposed a new motion planner that addresses the issue
when start state and goal state are not directly connectable
by sampling only in the discrete domain which shows a great
advantage over traditional motion planners especially in high di-
msional configuration space.

3 Preliminary

3.1 Nets of Polyhedra

The unfolding of a mesh can be obtained by finding a spanning
tree of the dual graph of the mesh [11]. Folding edges will be
those edges that are crossed by dual edges in the spanning tree.
All other edges will be cut to obtain the unfolding. If an unfold-
ing has no overlaps then we call it the net of the mesh.

3.2 Vertex Type and Local Overlaps

It is useful to classify vertices based on its Gaussian curvatures.
A hyperbolic vertex has negative Gaussian curvature and the sum
of the vertex angles of adjacent faces $S_t$ is greater than $2\pi$, and
an elliptic vertex has zero or positive Gaussian curvature and the
$S_t \leq 2\pi$. One cut on elliptic vertex’s adjacent edges is sufficient
to unfold that vertex without local overlaps. That is, all the ad-
jacent faces of that vertex are free of mutual overlap. Convex
polyhedron only contains elliptic vertices. On the contrary, non-
convex polyhedron contains hyperbolic vertices. At least two
cuts are required for each hyperbolic vertex in order to avoid
local overlaps in the unfolding. Because of these hyperbolic ver-
tices, heuristic methods developed for convex polyhedra which
find the ‘best’ cut edge for each vertex no longer works on non-
convex polyhedra.

3.3 Configuration Space of the Net

Given a mesh with $n + 1$ faces, its net (if exists) will have $n$
fold edges. We use $q = \{ p_1, p_2, \cdots, p_n \}$ to represent a folded
state of the net; $q$ is called a configuration, where $p_i \in [-\pi, \pi]$ is
the folding angle, i.e., $\pi - dihedral$ angle, of the $i$-th fold edge.
The configuration space $C$ is defined as the space of all possi-
ble configurations $\{ q \}$. Each point in $C$ represents a (partially)
folded state of the net. $C_{\text{obst}}$ is the space occupied by obsta-
cles in $C$. Since there is no real obstacle in the workspace
when we fold the net, the only ‘obstacle’ is the net itself as we
do not allow self-intersection during folding. We define
$C_{\text{obst}} = \{ q \in C \mid q \text{ is self-intersected} \}$ and then the free space
is defined as $C_{\text{free}} = C \setminus C_{\text{obst}}$. There are two special configura-
tions in $C_{\text{free}}$, $C_{\uparrow} = \{ 0, 0, \cdots, 0 \}$ and $C_{\downarrow} = \{ \theta_1, \theta_2, \cdots, \theta_n \}$ which
are the start state, i.e., the net, and the goal state, i.e., the original
Given a mesh \( M \), approach.

In this section, we will give an overview of the proposed approach.

Given a mesh \( M \), we first inflate the mesh to reduce surface concavity (Section 5), then remove structural concavity by decomposing the mesh into several part-aware, nearly convex components (Section 6). For each component we find a net using a genetic-based algorithm (GA) [21], the initial population are generated using heuristic methods. Once we obtained the net, motion planning (Section 7) is introduced to find a feasible path that folds the net back to its 3D shape continuously to ensure we can build a physical copy even use rigid materials instead of flexible materials, such as paper which could be easily bent during folding. Finally, all the components can be assembled. The pipeline of proposed approach is shown in Figure 1.

The main contributions of this paper are twofold: 1) Mesh inflation algorithms to reduce local concavity. 2) Validate the effectiveness of next convexity in unfolding/continuous folding.

5 Reduce Local Concavity via Mesh Inflation

Hyperbolic vertices are the main sources that cause existing unfolding methods fail to find a net. Because every hyperbolic vertex must be incident to at least two cuts, reducing the number of hyperbolic vertices implies the reduction in the variance of the number of cuts of each vertex and therefore simplifies the unfolding. Many of these hyperbolic vertices can be removed without affecting the overall shape. Inspired by physically based simulation to inflate a concave mesh into a balloon, We propose to use the idea to pop up small dents on the mesh and reduce the number of hyperbolic vertices so that the computation time of finding a net can be reduced. In particular, we will show that the net of an inflated model can be created with few modifications from an invalid unfolding generated by heuristic methods, which are usually designed for convex shapes.

5.1 Uniform (Unconstrained) Inflation

Force caused by air pressure should be uniformly distributed on the entire face, in this work, we simplify the model and assume forces only exist on the local region of each vertex \( v \). Then the force on the vertex is a weighted average of forces on adjacent faces,

\[
\vec{f}_p = \lambda_p p \sum \phi_i \vec{n}_i
\]

where \( \lambda_p \) is a coefficient, \( p \) is the pressure, \( \phi_i \) is the section angle of each adjacent face and \( \vec{n}_i \) is the normal direction of that face. Since \( \vec{f} = \vec{p}A = \vec{n}_i \vec{p}A \), at the local disk region around the vertex \( v \), \( \phi \) is proportional to \( A \), thus we can use \( \phi \) as the weight to compute the force contributed by each adjacent face.

During inflation, vertices moved, edges deformed. By assuming the mesh is made of elastic materials, the force applied to the vertex \( v \) due to stress on edge \( v_i v_j \) can be defined as,

\[
\vec{f}_{ij} = \frac{E_e \Delta L v_i v_j}{||v_i v_j||},
\]

where \( E_e \) is the elastic modulus of the material, \( \Delta L \) is the length changed of the edge defined as \( \Delta L = L_{cur} - L_{org} \). Since we would like to have the mesh inflated, only stretched case (\( \Delta L > 0 \)) is considered, otherwise \( f_{ij} = 0 \).

The total elastic force on vertex \( v \) is the summation of forces on adjacent edges,

\[
\vec{f}_e = \sum_{j} \vec{f}_{ij}.
\]

The total force on each vertex is then,

\[
\vec{f} = \vec{f}_p + \vec{f}_e.
\]

The position \( \vec{P} \) of the vertex is updated in an iterative manner and we assume each vertex has the same unit mass and zero mass elsewhere,

\[
\vec{P}_{t+1} = \vec{P}_t + \Delta t \vec{f}.
\]

5.2 Constrained Inflation

Although uniform inflation works well in practice in terms of reducing the number of hyperbolic vertices, it introduces several undesired properties for our application: (1) convex region will be inflated and (2) flat surfaces cannot be maintained. Consequently, the inflated shape loses the structural resemblance of the
In order to maintain convex regions and keep deep concave regions, we multiply a stiffness ratio to the edges based on their current folding angle,

\[
\vec{f}_e' = \begin{cases} 
\lambda_{se} \vec{f}_e, & \text{edge is convex or deep concave} \\
\vec{f}_e, & \text{otherwise}
\end{cases} \tag{6}
\]

where \(\lambda_{se}\) is the stiffness ratio coefficient.

**Virtual Edges** We add virtual elastic edges between the current position of the vertex \(P'\) and its original position \(P\) to penalize large displacement of the elliptic vertex. The pulling force applied to the vertex is similar to the one shown in Eq. (2).

\[
\vec{f}_v = \frac{\lambda_v E_v \Delta L_{PP'}}{||PP'||} \tag{7}
\]

\[
\lambda_v = \begin{cases} 
1, & \text{hyperbolic vertex} \\
\lambda_{sv}, & \text{elliptic vertex}
\end{cases} \tag{8}
\]

where \(\lambda_{sv}\) is another stiffness ratio coefficient.

Finally, Eq. (4) can be rewritten as,

\[
\vec{f} = \vec{f}_p + \vec{f}_e' + \vec{f}_v \tag{9}
\]

5.3 Experimental Results

An example of mesh inflation can be found in Figure 3, from which we can see that inflation can reduce the number of hyperbolic vertices effectively. However without any constraints, the mesh will be inflated to a sphere-like shape finally (if the pressure is high enough). Though all the concave features were removed, the volume of the mesh increased radically. Constrained inflation helps to achieve similar results with much less inflation. We also measure the unfolding time, folding time and folding success rate on original mesh and inflated ones. From Figure 3 we can see that inflation removes hyperbolic vertices which helps GA to find the net more efficiently and more effectively. It also helps to find the feasible folding path more efficiently and more effectively. The reported unfolding time is the average of 30 runs on each inflated mesh; path planning time is the average of 20 runs on each found net; time limit for each trail is 600 seconds. And we can see that constrained inflation achieves similar or better results with much less deformation compared to uniform inflation.
In this paper we employ a method called Convex Ridge Separation (CoRiSe) \cite{10} to decompose a 3D mesh to part-aware with controllable concavity of all components in the decomposition. The concavity of a shape is defined as the maximum distance between the convex hull and the shape. It is important to note that other segmentation methods can also be incorporated with the proposed framework as long as the convexity of the compo-
ment can be bounded. For example, an alternative approach can use Continuous Visibility Feature \[22\] to repetitively segment the shape until certain desired convexity is reached.

7 Continuous Folding of Nets

Once obtained the net, the next question we want to address is whether there exists a continuous folding motion that transforms the net back to the original mesh. Since the net has a very high degree of freedom which equals to the \(|F| - 1\) where \(|F|\) is the number of faces, which could be hundreds or even thousands that makes traditional motion planners failed to work. We employ the method from \[9\] to plan continuous folding motion for the net.

7.1 Sampling in Discrete Domain

The net usually has a high DOF, sampling in the continuous domain has extremely low probability (close to 0) to generate a valid configuration which is known as the curse of dimensionality that makes traditional motion planners failed to find any path. Instead, by only sampling in discrete domain that the folding angle of a crease line is sampled from \(\{0, \theta_i\}\) where \(\theta_i\) is the target folding angle of that fold edge. There are \(2^{|F| - 1}\) possible states in the discrete configuration space. Figure 5 shows the valid configuration ratios defined as \(|C_{valid}|/|C_{sampled}|\) of various models and its nearly convex decomposition components whose DOFs are ranging from 7 to 479 for two sampling strategies, from which we can see that sampling in discrete domain maintains a much higher probability, which is exponentially higher than uniform sampling. Generally speaking, the higher the valid configuration ratio is the easier to find a feasible path in the C-space. Also, we can see that for discrete domain sampling, nearly convex decomposed patches have a higher valid configuration ratio compare to that of original mesh which implies they are easier to fold.

7.2 Plan the Motion for the Net

Lazy-PRM \[23\] is employed to find feasible folding paths. The motion planner starts with sampling certain amount of configurations in the discrete domain and adds valid ones, those without self-intersection, to the roadmap. Then the planner tries to connect nearby configurations. Finally a graph search is performed to find a path from the start state to the goal state. This process can be repeated until a path is found or time limit was exceeded. We say two valid configurations are directed connected (there will be an edge between them in the roadmap) if there is a linear path between them that every point on the path is self-intersection.

FIGURE 5. Valid configuration ratios of various original models (Org) and their components decomposed by nearly convex decomposition (NCD) under two sampling strategies.

Patches generated from nearly convex decomposition have a much higher probability that a straight line from state to goal in the high dimensional C-Space is a feasible folding path which can be tested in \(O(|F|^2)\) with a naive collision detection method. If start state and goal state can not be directed connected then it may take up to \(O(n^2|F|^2)\) to find a path or report a failure, where \(n\) is the maximum number of valid samples. The huge path planning time differences between straight line paths and non-straight line paths are shown in Table 1 and Table 2. We encourage readers to visit our web-based interactive folding process visualizer to experience the complexity and beauty of the folding process.

7.3 Experimental Results

7.3.1 Experiment Setup We implemented the proposed unfolding/folding methods in C++. All data reported in this paper were collected on a 2012 MacBook Pro with a 2.9 GHz Intel Core i7 CPU and 16GB Memory running Mac OS X. Meshes used in the experiment are shown in Figure 6 as their decomposed states, components are shown in different colors.

7.3.2 Running Time We compare the running time of unfolding the entire mesh and sum of running time of unfolding the decomposed components. The results (average running time of 20 runs) are shown in Table 3 from which we can see that nearly convex decomposition can significantly reduce the total running time especially on path planning, and the extra running time introduced by CoRiSe is negligible. For path planning, the main reason that nearly convex decomposition helps is that a convex shape usually has a good net that the start and goal are directly linear connectable \[9, 17\]. Thus we do not even need to plan
a path, but only need to validate it. And, as we can expect, nearly convex patch should have similar properties which will make continuous folding much easier. For the decomposition results of Periscope and Periscope2, all of their decomposed components have a linear path that directly connects start and goal, which significantly reduced the total running time (see Table 2).

For the fish model shown in Figure 6(d) after nearly convex decomposition, the largest component (the body) still has 333 faces (original mesh has 474 faces). However, since all the components are nearly convex, both total running times (the summation of all components) reduced significantly especially for finding the continuous motion, which is 16.12 times faster before inflation and 163.56 times faster after inflation.

### Table 1. Running Time of the Horse Model (s)

<table>
<thead>
<tr>
<th>Patch</th>
<th>DOF</th>
<th>LC</th>
<th>Finding Net</th>
<th>Path Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17</td>
<td>✓</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>1</td>
<td>17</td>
<td>✓</td>
<td>0.00</td>
<td>0.02</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>×</td>
<td>0.00</td>
<td>2.69</td>
</tr>
<tr>
<td>3</td>
<td>42</td>
<td>×</td>
<td>0.56</td>
<td>1.39</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>×</td>
<td>0.02</td>
<td>0.73</td>
</tr>
<tr>
<td>5</td>
<td>86</td>
<td>✓</td>
<td>4.84</td>
<td>0.31</td>
</tr>
<tr>
<td>6</td>
<td>75</td>
<td>×</td>
<td>0.58</td>
<td>1.60</td>
</tr>
<tr>
<td>Sub Total</td>
<td>6.00</td>
<td></td>
<td>6.75</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>12.75</td>
<td></td>
</tr>
</tbody>
</table>

LC: Whether $\theta_s$ and $\theta_f$ are linear connectable in $\theta_{free}$.

For the horse model shown in Figure 6(e) we obtained the net for the original mesh which takes 219.87 seconds, however, we were not able to find a feasible folding path for the net in a certain amount of time (>1 hour). Using the proposed idea, we decompose the horse mesh to 7 part-aware and nearly convex components, in which, 4 for the legs, 1 for the head, 1 for the neck and 1 for the body. We then find the net and the folding path for each component separately. Average running time of 20 trails of each patch is shown in Table 1 from which we can see that nearly convex patch is more likely to be unfolded to a net that the start and goal can be linearly connected, thus the path planning time can be reduced significantly even in very high dimensional space. One observation worth noting is that since we can find different nets for each component, the shape and the topology of the nets, which determine whether start and goal are linearly connectable, have a huge impact on the difficulty of finding a feasible folding path. How to measure the quality of a net in terms of its foldability is an interesting question and remains as future work.

### 8 Physical Models

In order to illustrate the proposed method, we built several physical models based on the nets generated by the proposed method. We export each net to a SVG (Scalable Vector Graphics) file in which the boundary of the net is represented as a polygon and all the crease lines are represented as single paths. The SVG file is inputted to Cricut Explore® machine which will cut the polygon (boundary of the net) out of the paper (12in × 12in) and score all the crease lines which makes folding much easier. We then manually fold each component and stitch all the cutting edges using types. Final results are shown in Figure 7.

### 9 Conclusion

Edge unfolding a polyhedron to a single or multiple non-overlapping connected pieces (i.e. nets) and planning folding motion that continuously transforms every piece back to original polyhedron are two main steps in designing and manufacturing foldable shapes. In this paper, we proposed the first known method that applies mesh processing methods to increase the possibility of generating successful edge unfolded nets and folding motion. The proposed method first inflates the mesh to remove local concave features, then employs nearly convex de-
TABLE 2. Running Time (s)

<table>
<thead>
<tr>
<th>Model</th>
<th># of Parts</th>
<th>DOF</th>
<th>CoRiSe</th>
<th>Finding Net</th>
<th>Path Planning</th>
<th>Total</th>
<th>Speedup (X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periscope</td>
<td>1</td>
<td>27</td>
<td>-</td>
<td>0.08</td>
<td>7.05</td>
<td>7.13</td>
<td>59.33</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>13 / 13</td>
<td>0.00</td>
<td>0.00</td>
<td>0.12</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>Periscope2</td>
<td>1</td>
<td>43</td>
<td>-</td>
<td>0.21</td>
<td>12.40</td>
<td>12.61</td>
<td>48.50</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>13 / 15 / 13</td>
<td>0.00</td>
<td>0.09</td>
<td>0.17</td>
<td>0.26</td>
<td></td>
</tr>
<tr>
<td>Bunny</td>
<td>1</td>
<td>127</td>
<td>-</td>
<td>8.09</td>
<td>482.57</td>
<td>490.66</td>
<td>18.82</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4 / 6 / 115</td>
<td>0.02</td>
<td>2.90</td>
<td>23.15</td>
<td>26.07</td>
<td></td>
</tr>
<tr>
<td>Fish</td>
<td>1</td>
<td>473</td>
<td>-</td>
<td>69.66</td>
<td>1848.86</td>
<td>1918.52</td>
<td>16.12</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>26 / 34 / 36 / 44 / 332</td>
<td>0.92</td>
<td>13.19</td>
<td>105.82</td>
<td>119.01</td>
<td></td>
</tr>
<tr>
<td>Fish-inflated</td>
<td>5</td>
<td>26 / 34 / 36 / 44 / 332</td>
<td>0.92</td>
<td>0.88</td>
<td>9.93</td>
<td>11.73</td>
<td>163.56</td>
</tr>
</tbody>
</table>

FIGURE 7. Physical models built using the proposed method.

composition to segment the original mesh to a small number of part-aware and nearly convex patches and solve both unfolding and folding problems on each patch separately which shows great advantages in both finding nets and feasible folding paths as well as manufacturing and assembling.

Limitations and Future Works Though our preliminary results are promising, we found it is still a challenging problem when the polyhedron has a large number of faces and concave features. In addition to inflation and segmentation, we are currently investigating alternative mesh processing methods, such as remeshing [24] and denoising [25]. In addition, it may take a very long time for GA to find a solution and it is difficult to add stop criteria when there is no solution. We also noticed that for a given mesh, path planning time is heavily dependent on the nets. How to define a goodness of a net and integrate that into the GA fitness function remains as the future work.

Acknowledgement

This work was supported in part by NSF IIS-096053, CNS-1205260, EFRI-1240459, AFOSR FA9550-12-1-0238.

REFERENCES


